Developer Guide

June 2013 Edition

Worldwide technical support and product information:

www.toolsforsmartminds.com

TOOLS for SMART MINDS Corporate headquarter

Via Padania, 16 Castel Mella 25030 Brescia (Italy)

Copyright © 2012 TOOLS for SMART MINDS. All rights reserved.

CONTENTS

ABOUT THIS MANUAL **8**

CONVENTIONS 8

PREFACE **10**

REQUIREMENTS **10**

MINIMUM HARDWARE REQUIREMENTS 10

MINIMUM SOFTWARE REQUIREMENTS 10

INSTALLATION **11**

GETTING STARTED WITH IDAQ-DT **12**

CONNECTING WITH DATA SOURCE 12

Essential connection parameters and functions *12*

Data source selection *13*

DISCONNECTING FROM DATA SERVER 15

RECEIVING DATA 16

Starting data communication *16*

Pausing data communication *17*

CREATION OF A SIMPLE APPLICATION STAND-ALONE **18**

Creating the data displaying while loop *19*

Creating the event handler while loop *20*

ANALOG SIGNALS FILTERING **22**

Default data transmission *22*

Filtering process and functions *23*

Signal filtering: a practical example *24*

CREATING ADD-ONS FOR IDAQ **26**

ADD-ONs requirements *26*

Getting started with ADD-ONs development *27*

APPENDIX A: USEFUL LINKS **34**

FIGURE INDEX 36

INDEX **39**

About this Manual

Conventions

The following conventions appear in this manual:

▸	The ▸ symbol leads you through nested menu items and dialog box options to a final action. The sequence **Tools** ▸ **Options** directs you to pull down the **Tools** menu, select **Options** item.
Bold	Bold text denotes items that you must select or click on the software, such as menu items and dialog box options. Bold text also denotes parameter names.
italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.
`monospace`	Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excerpts.

`monospace italic`

Italic text in this font denotes text that is a placeholder for a word or value that you must supply.

PREFACE

iDaq–Development Toolkit (iDaq-DT) is a development toolkit to create standalone clients capable to connect to iDaq servers and to create ADD-ONs fully integrate into iDaq servers.

REQUIREMENTS

In this section are listed requirements that users must have in order to install and use iDaq-DT properly. It is strongly recommended to check them out before starting iDaq-DT installation.

MINIMUM HARDWARE REQUIREMENTS

Ensure that your computer fulfill the following hardware requirements:

- Dual core CPU or higher
- 2 GB RAM (4 GB is strongly recommended)
- Display resolution 1024x768 or higher
- 200 MB free space on disk required by iDaq
- about 800 MB free space on disk required by National Instruments device drivers
- mouse or other pointing device

MINIMUM SOFTWARE REQUIREMENTS

Ensure that your computer fulfill the following software requirements:

- OS: Windows XP/SP3 or higher.
- National Instruments device drivers rel 9.3.x or higher. You can download a free copy of the proper device driver for your acquisition device at www.ni.com.
- LabVIEW 2011 to create Standalone applications.
- LabVIEW 2012 to create both Standalone applications and ADD-ONs for iDaq.
- SCCT (Smart phone & Cross-platform Communication Toolkit) for LabVIEW 2.6.x version or higher, downloadable at http://www.toolsforsmartminds.com/products/SCCT.php . If you need any information about SCCT, you can find it in its user manual available at http://www.toolsforsmartminds.com/pdf/SCCT%20User%20Manual.pdf .
- iDaq: you can find out all the information about iDaq installation in its user manual, available at http://www.toolsforsmartminds.com/pdf/iDaq%20User%20Manual.pdf .

INSTALLATION

In order to install iDaq-DT, since it is a VIP package, first you need to download VIP Manager from JKI. You can download a free copy at:

https://lumen.ni.com/nicif/us/evaltlktvipm/content.xhtml

Once you have downloaded VIP Manager, you can proceed with iDaq-DT installation. You can find it at :

www.toolsforsmartminds.com/products/programmable_data_logger.php .

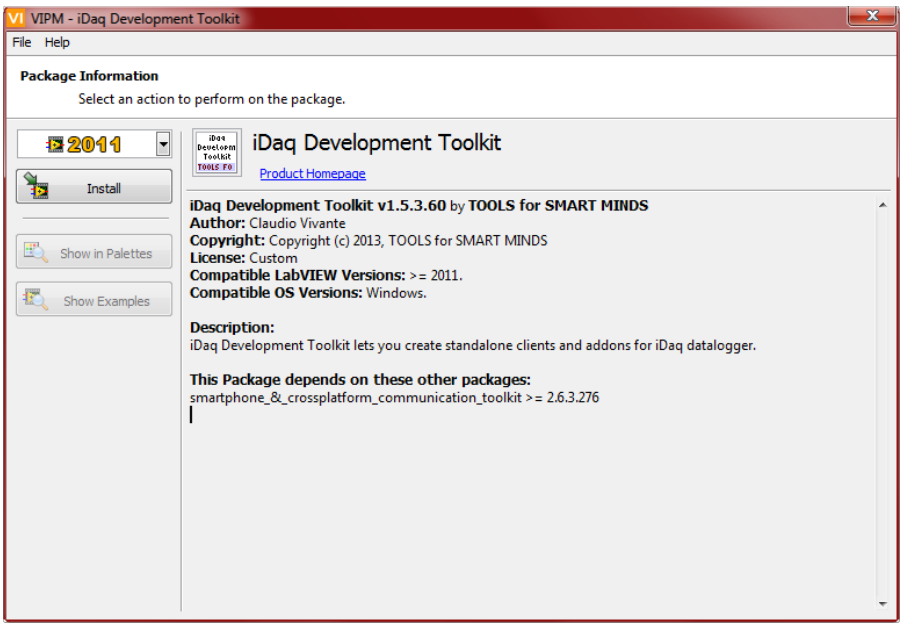
VIPM - iDaq Development Toolkit
File Help
Package Information
Select an action to perform on the package.
2011
Install
Show in Palettes
Show Examples
iDaq Development Toolkit
Product Homepage
iDaq Development Toolkit v1.5.3.60 by TOOLS for SMART MINDS
Author: Claudio Vivante
Copyright: Copyright (c) 2013, TOOLS for SMART MINDS
License: Custom
Compatible LabVIEW Versions: >= 2011.
Compatible OS Versions: Windows.
Description:
iDaq Development Toolkit lets you create standalone clients and addons for iDaq datalogger.
This Package depends on these other packages:
smartphone_&_crossplatform_communication_toolkit >= 2.6.3.276

FIGURE 1 IDAQ-DT INFORMATION PACKAGE DURING GUIDED INSTALLATION.

GETTING STARTED WITH IDAQ-DT

This chapters introduce you to the main features of iDaq-DT and on how much their implementation can be useful and high-performing once matched with iDaq and SCCT.

Since the purpose of this manual is leading and supporting you step by step through iDaq-DT functionalities in the simplest way, every chapter provides you with a brief introduction, accurate indications and useful images and diagrams which show you how processes actually work.

CONNECTING WITH DATA SOURCE

This chapter starts on explaining about how you can open a connection with data server: the just mentioned data servers are instances of iDaq being run on local or remote machines.

First of all, therefore, you have to enable iDaq to open remote connections: you can do it easily by clicking on the menu **Options ▸ Data distribution**. In order to connect, you must know connection credentials and data server IP address.

ESSENTIAL CONNECTION PARAMETERS AND FUNCTIONS

iDaq-DT is made up of a set of functionalities based on particular objects called `iDaqTask`. In order to be used, `iDaqTask` objects (hereinafter, for the sake of brevity, they are simply called Tasks) need to be initialized through a specific function called `startTask.vi`.

The next figure shows the LabVIEW code needed for opening connection with a server.

FIGURE 2 NECESSARY CODE FOR SERVER CONNECTION

The `credentialFilePath` parameter you can see in the figure before, serves the purpose of specifying the absolute path of the credential file you need to use. The initialization of a credential file allows at the same time to verify its validity. In case that `credentialFilePath` hasn't been specified, `StartTask.vi` demands users to create a new credential file and, through a dedicated editor, allows data server to verify the connection.

The next figure shows the connection parameters editor:

FIGURE 3 CONNECTION PARAMETERS EDITOR

The following list resumes and specifies the credential values needed to open a new connection:

- **Server address:** localhost
- **Server port:** 9000
- **Server password:** iDaq

You can verify if the credentials you've typed in are actually right for the data server by means of the **Test** button, then click on **OK** button to save connection parameters.

Once `iDaqTask` object has been initialized, you can finally open the communication with data server. In order to connect `iDaqTask` object, apply `connect.vi` function as suggested in the next figure:

FIGURE 4 CONNECTING IDAQTASK WITH CONNECT.VI

If data server is either not found or not accessible or used credentials have been rejected, `connect.vi` returns an error. You can easily identify what kind of error is it by implementing the `error out.source` parameter.

DATA SOURCE SELECTION

When connection has been successfully established, data server transmits immediately information about all the available acquisition devices (e.g. data sources), in order to handle in the best way the communication. In facts, if for example server data sources turn out to be dynamic (that is, they change as time passes by) the server updates data about available sources characteristics.

Getting information about the available data sources is very easy: you simply have to implement the `Read sourceList.vi` function, as explained in the following figure:

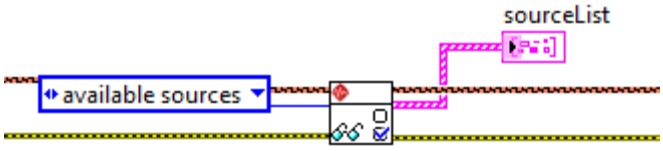

FIGURE 5 GETTING DATA SOURCE INFORMATION W ITH READ SOURCELIST.VI FUNCTION

In order to get, instead, the available data sources characteristics, use the `Read signalTable.vi` function. Every data source is described by a cluster, and `Read signalTable.vi` actually returns an array of clusters which allows you to get all the details you need.

NOTE:

Remember to elaborate cluster data only when the Boolean index `found` returns a TRUE value, as indicated in the next figure:

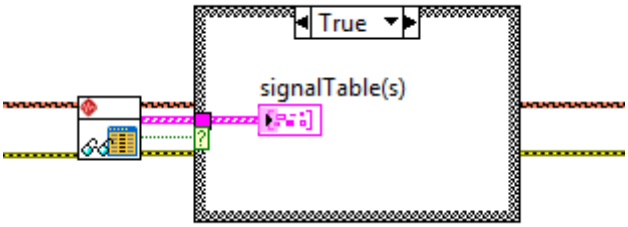

FIGURE 6 ELABORATING CLUSTER WHEN BOOLEAN FOUND = TRUE

`connect.vi` function picks automatically the first data source available if no data source has been selected before. If you want to connects to different sources, use Changesource.vi. An iDaq task can be connected to one or more data sources. After sources are changes, iDaq sends all data acquired from selected sources to your task. In the following figure, an example in which two sources (“Dev2” e “Dev5”) are being selected is shown.

FIGURE 7 DATA SOURCE SELECTION

Use the comma character “,” to separate the selected sources names.

To select data source, you can use the function `SourceSelectionDialog.vi` which provides user with a list of all the available data sources and their characteristics and which automatically make the selection. `SourceSelectionDialog.vi` allows you to select one or more sources. In the following figure you can see a sample code for source selection and the corresponding dialog box:

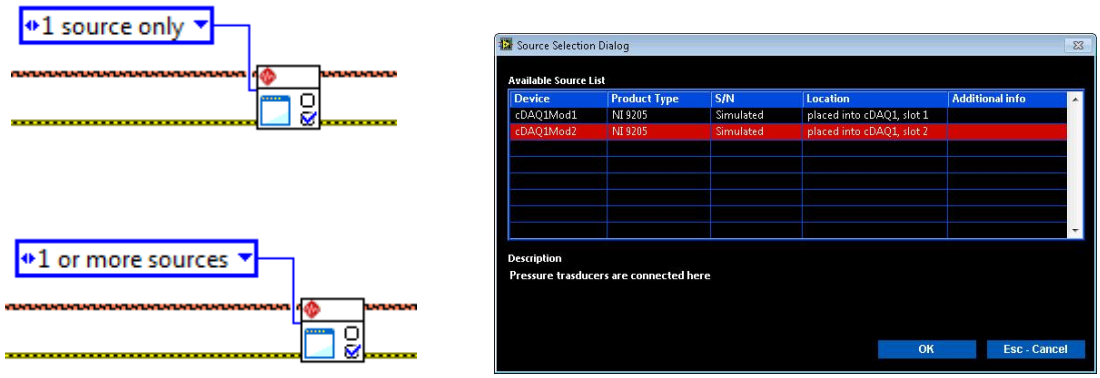

FIGURE 8 SOURCE SELECTION SAMPLE CODE AND DIALOG BOX

This chapter goes on explaining about the matter of data servers connection: it analyzes in particular how to close communication with data servers, gives useful indications about the functions iDaq-DT uses and how to handle them.

In order to disconnect from a data server, implement the `disconnect` function as indicated in the next example:

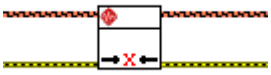

FIGURE 9 DISCONNECT FUNCTION

NOTE:

If a client closes communication without having been disconnected before with `disconnect.vi`, the server can determine anyway if that client isn't connected anymore and therefore automatically closes the connection. It's a good rule, however, to use `disconnect.vi` command at the end of every connection in order to allow `iDaqTask` to clear the allocated memory and to terminate background tasks.

RECEIVING DATA

After having explained all the details about how to open or close communications with data servers, this chapter examines in depth the process of receiving data after that they have been sent from different data sources. Within this chapter you will find how the entire receiving process has been built and learn about the functions iDaq-DT gives you to achieve the best performances.

Client receives data through the `Read streams.vi` function, that includes the timing of the loop in which this function has to be integrated. `Read streams.vi` returns a packet of analog and digital data equipped of identification details of the data source from which it was generated.

The figure below shows a while loop in which packets concerning data streams are being real time extracted from the data source queue.

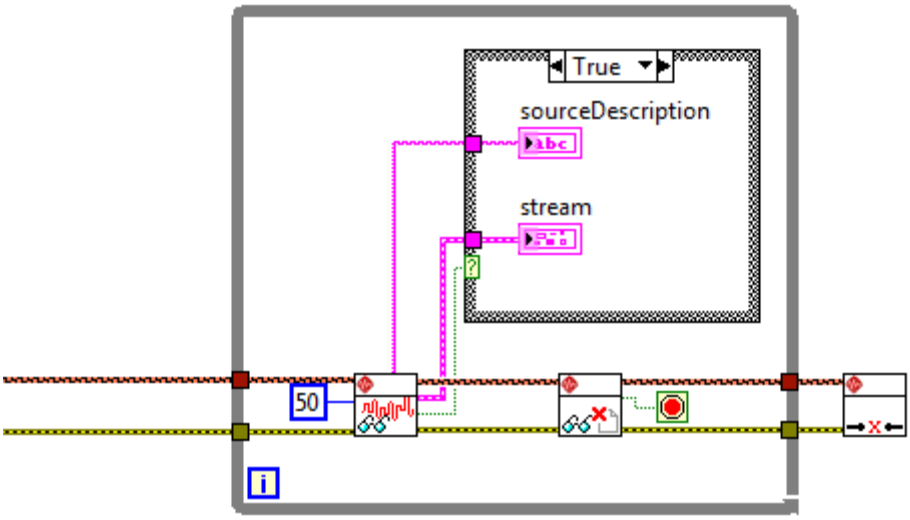

FIGURE 10 REAL TIME EXTRACTING PACKETS IN WHILE LOOP

Data server configures conveniently its own routines according to the source selected by the client, but it is also possible to change the data source selection in any moment during the server connection. The client can change data source repeatedly during the connection.

STARTING DATA COMMUNICATION

After having been specified the name of the source from which you want to receive data, the client has to inform the server that it's ready to receive data. In order to start data transmission from the source, you can use the `play.vi` function, as shown in the following figure:

FIGURE 11 STARTING DATA SOURCE TRANSMISSION WITH PLAY.VI

PAUSING DATA COMMUNICATION

When you have received all the data you need, you can temporarily stop data communication from server side, using the `pause.vi` function as explained in the next figure:

FIGURE 12 STOPPING DATA SOURCE TRANSMISSION WITH PAUSE.VI

The client can pause and restart data transmission again and again until server is connected. When the client uses the pause.vi command, the server automatically rejects the not sent data packets, therefore the client can't receive them after having restarted data communication. The next diagram explains the data transmission sequence between server and client.

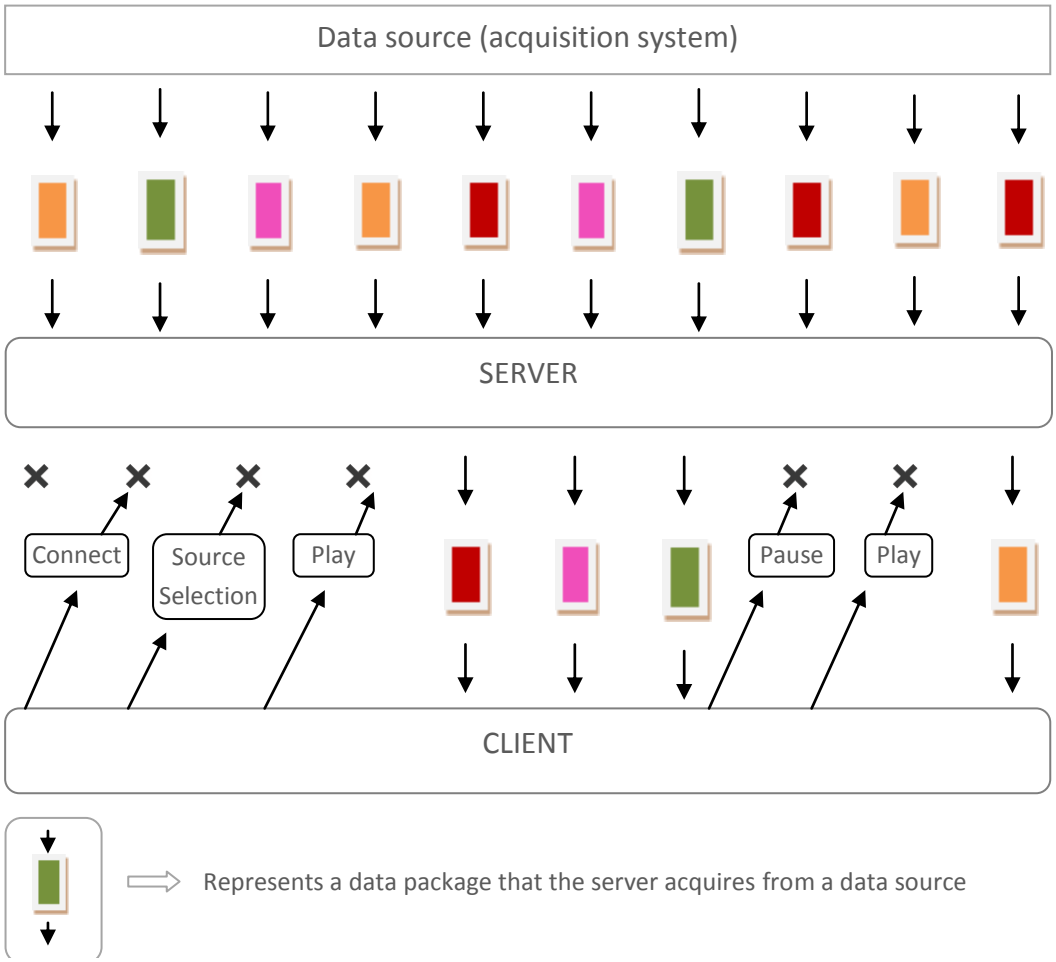

FIGURE 13 DATA TRANSMISSION PROCESS FROM DATA SOURCE TO CLIENT,

CREATION OF A SIMPLE APPLICATION STAND-ALONE

This chapter will guide you step by step in the creation of a sample basic application that you can easily realize thanks to iDaq Development Toolkit. The application is programmed to have the following features:

- Establish a connection with a data server;
- Allow to select a data source among the available ones;
- Display the chart of the first available analog channel .

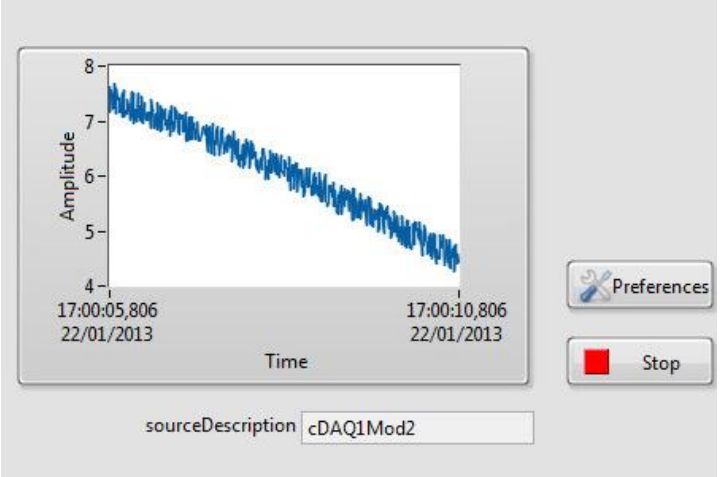

FIGURE 14 SAMPLE APPLICATION OVERVIEW

The complete source code of this project is available at the following link: www.toolsforsmartminds.com

The application is based on two loops:

1. **The first while loop** waits for the analog data packets coming from the data source and displays the chart of the first analog channel among those available.
2. **The second loop** has the task of handling events and therefore of interacting with users.

The user is provided with a **"Stop" button** to stop the client and a **"Change source" button** to select a data source.

The client, at the startup of the process, uses a file with connection credentials: if this file isn't available it is possible to create a new one with these credentials.

NOTE:

Generally iDaq developers use the .SCCT format for files containing connection credentials.

The following figure illustrates the code you have to use to upload the credential file, to initialize the `iDaqTask` object and the server connection.

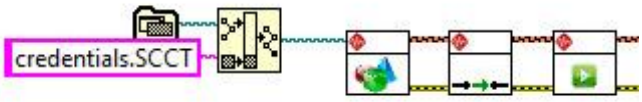

FIGURE 15 SAMPLE CODE TO INITIALIZE IDAQTASK AND SERVER CONNECTION

CREATING THE DATA DISPLAYING WHILE LOOP

As just said before, **this first while loop has the task of interacting with user**, and contains the following events: `"stop": Value change` and `"changeSource": Value change`.

In the image that follows, you can see that there are being used two functions of iDaq-DT:

1. `Read Streams.vi:` to get the next data packet and display the first analog signal;
2. `isTerminated.vi`: it returns TRUE when the `Task` has been terminated.

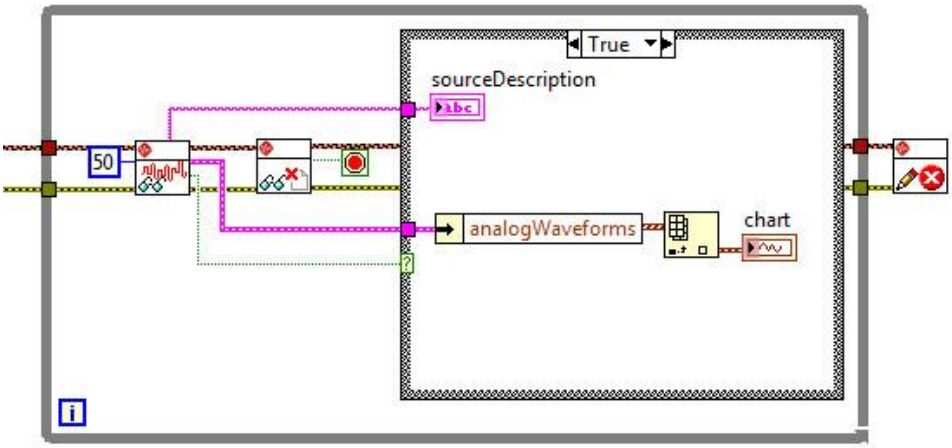

FIGURE 16 DATA ACQUISITION WHILE LOOP

NOTE:

There are many ways to end the loops of a LabVIEW applications and every developer chooses its own favorite. For iDaq-DT applications it's suggested to use isTerminated.vi and isTerminated.vi functions to verify the task status and to end the activities regarding connections with the iDaq server.

CREATING THE EVENT HANDLER WHILE LOOP

The second while loop, which is the event handler, is made up of the classical LabVIEW structure for event handling and mainly it handles two events:

1. **The first event**, linked to the **`stop` button**, is necessary to stop the data visualization loop described before. The `terminate.vi` function closes all the activities managed by the `Task` object. The following image resumes how the event handling `stop` button works.

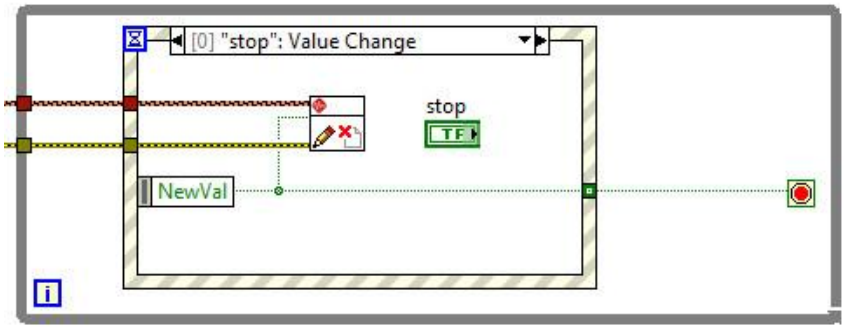

FIGURE 17 SAMPLE CODE OF AN EVENT HANDLING STOP BUTTON

2. **The second event handles the source selection**. `iDaqTask` object has a particular procedure of private data handling that doesn't require local variables applying. This is the reason why the implementation of the `selectSourceDialog.vi` function actually doesn't refresh the `Task` object after the execution of `sourceSelectionDialog.vi` function. The next image exemplifies the code needed to create a source selection event:

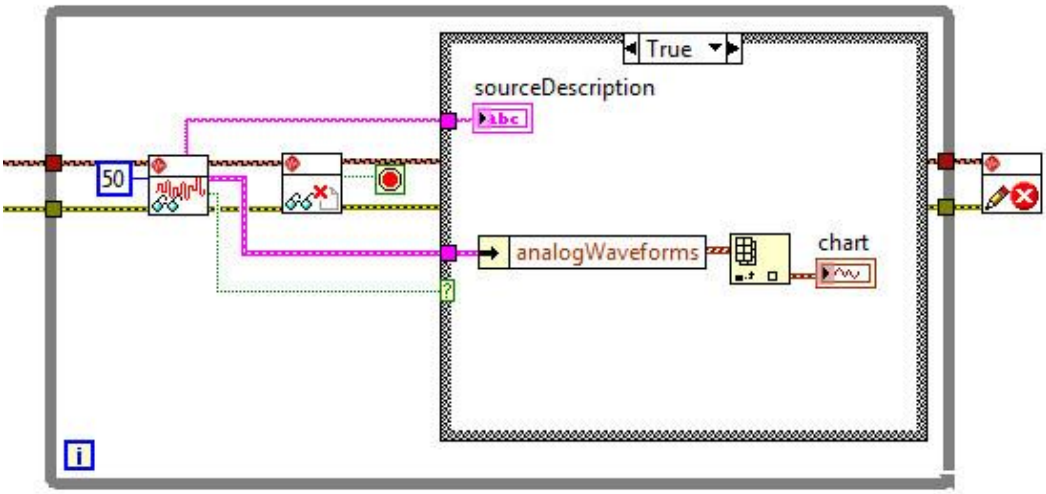

FIGURE 18 SAMPLE CODE OF AN EVENT HANDLING SOURCE SELECTION

It is very important, at the end of the data displaying loop, to use the `disconnect.vi` function, in order to advise the server that the connection has been closed and to remove data used by iDaq-DT from the memory.

In the next figure you can finally see the resulting application diagram:

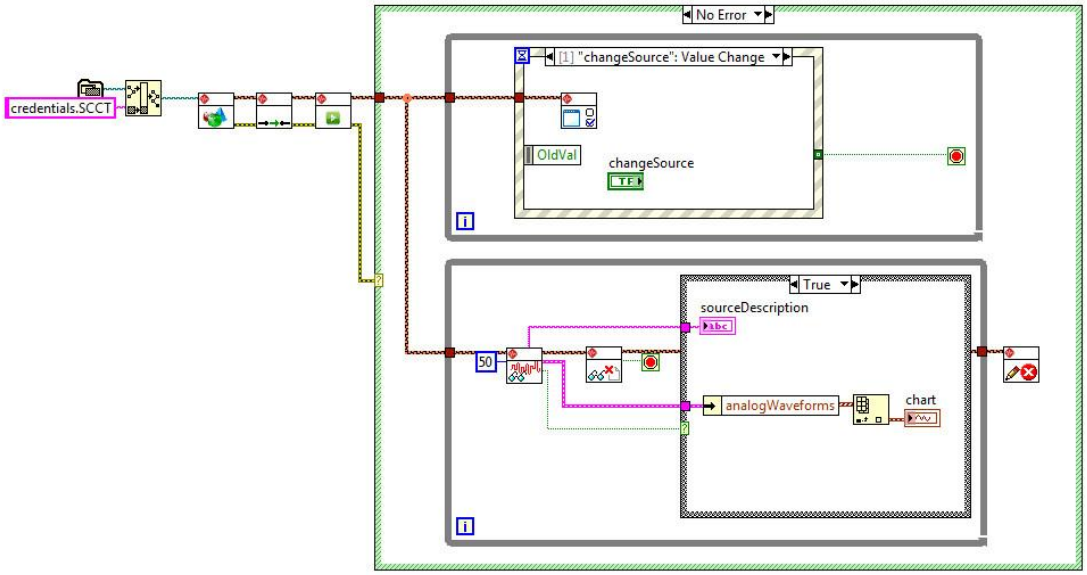

FIGURE 19 RESULTING CODE OF A SAMPLE STAND-ALONE APPLICATION

ANALOG SIGNALS FILTERING

This chapter describes the advanced iDaq-DT functions that allow to optimize CPU use as regards the client and to greatly reduce band wasting of the communication channel.

NOTE:

All the functions explained in this chapter are fundamental for all those who develop applications intended to be used in locations equipped with low bandwidth communication channel, for example Wi-Fi connections and some types of internet connections.

DEFAULT DATA TRANSMISSION

Data transmission process from iDaq server to a client is precisely described by the next diagram:

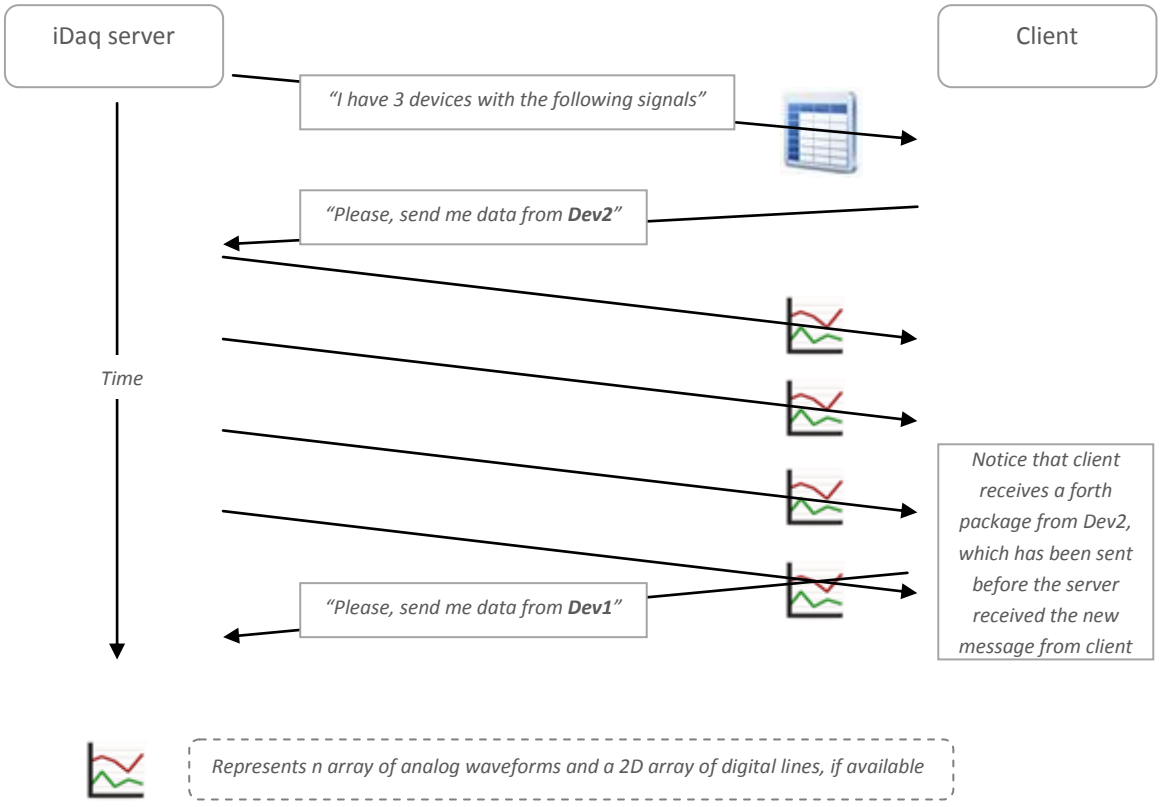
iDaq server
Client
"I have 3 devices with the following signals"
"Please, send me data from Dev2"
Time
Notice that client receives a forth package from Dev2, which has been sent before the server received the new message from client
"Please, send me data from Dev1"
Represents n array of analog waveforms and a 2D array of digital lines, if available

FIGURE 20 SERVER-CLIENT DATA TRANSMISSION PROCESS DIAGRAM

By default iDaq server sends to every channel of the selected data source. If you want to configure a source, we suggest you to take a look at the **iDaq user manual**: you can find its link in the **Appendix A: useful links** at the end of this manual.

FILTERING PROCESS AND FUNCTIONS

When a client needs to use a specific subset of the analog channels available in data source, **it is possible to require the server to send only those selected channels.**

EXAMPLE:

- *A data source has available 16 analog channels.*
- *A client has to use, for a specific activity, only channels 2, 3, 7 and 11.*

The client can require the server to send only the needed analog channels waveforms, thanks to the `filterData.vi` function. The next figure represents how to implement this channel selecting function:

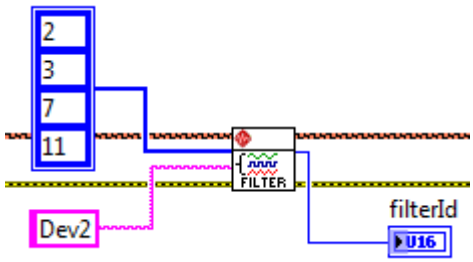

FIGURE 21 SAMPLE CODE FOR FILTERDATA.VI IMPLEMENTATION

`filterData.vi` function returns a number called FilterID. This number identifies the filtering request just sent to the server and it increases at every following request of channel selection by client.

`Read streams.vi` function returns ID number of the applied filter by data server to each sent data packet. Comparing the ID number returned by Read streams.vi with the ID number returned by `filterData.vi` is fundamental to identify the signals coming from server accurately. A different FilterId is generated for every filter request. When a new filtering operation is performed, FilterID is incremented. The FilterID permits the client to relate every received data package to the relative filter request. FilterId is fundamental to process received data in the right way. It is also possible to delegate the control of `FilterID` to `iDaqTask` object, so that only the iDaq packets provided with corresponding `FilterID` can be accepted. If you want to give `iDaqTask` the duty of handling filters, you simply have to use the `matchFilter.vi` function with parameter = TRUE, as indicated in the next figure.

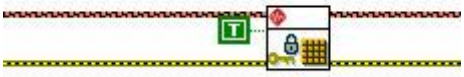

FIGURE 22 DELEGATING FILTERID CONTROL TO IDAQTASK

SIGNAL FILTERING: A PRACTICAL EXAMPLE

The next diagram shows a sequence of packets sent by server to client when the latter uses the `filterData.vi` function.

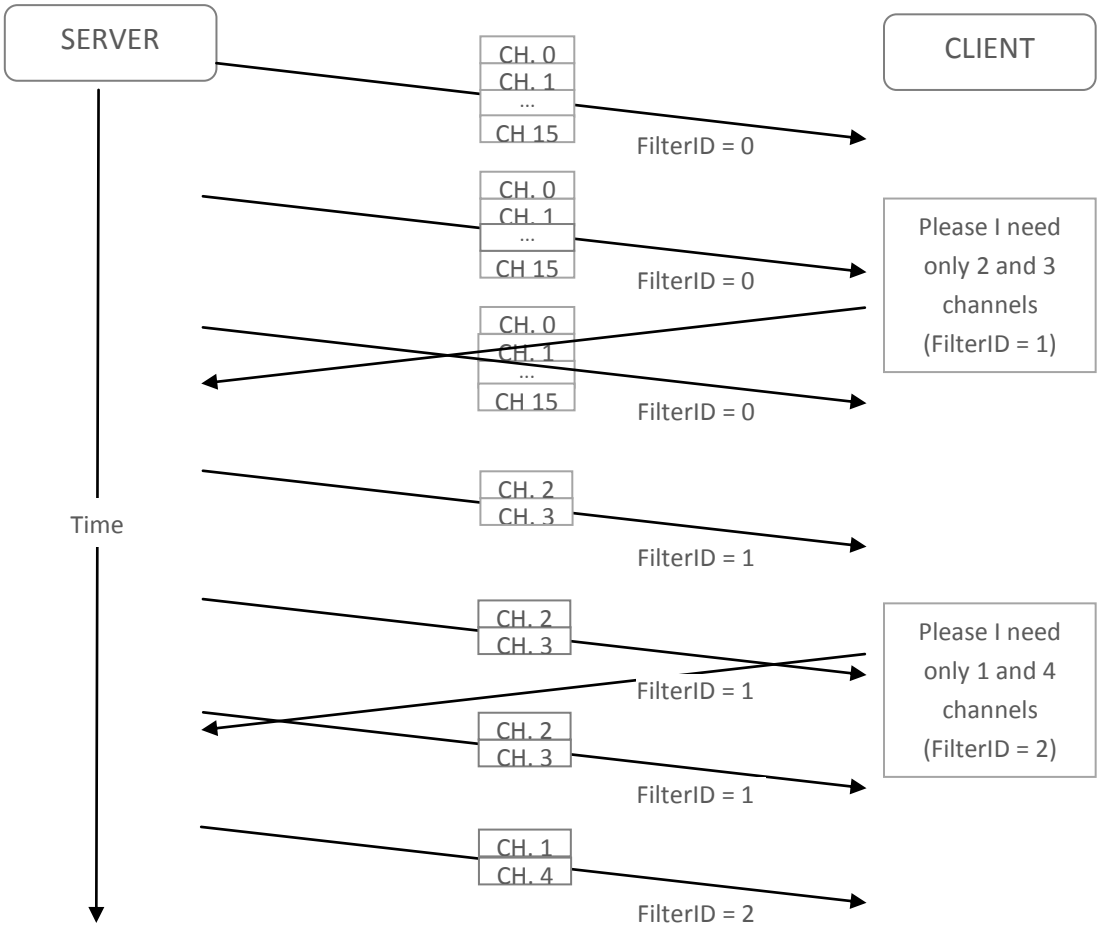

FIGURE 23 SEQUENCE OF PACKET EXCHANGED BY SERVER AND CLIENT WITH FILTERDATA.VI

In this diagram the client calls for channels 2 and 3 and then for channels 1 and 4: `FilterID` allows the client to distinguish which packet with two waveforms actually contains signals either of channels 2 and 3 or of channels 1 and 4.

CREATING ADD-ONS FOR IDAQ

This chapter explains how to develop useful ADD-ONs fitting to all the needs: thanks to iDaq-DT, in facts, you can finally create personalized ADD-ONs for iDaq servers.

Using ADD-ONs allows you to take advantage of certain functionalities indeed not developable through stand-alone applications at all, for example:

- Complete integration in iDaq interface;
- Inclusion of acquiring tasks into iDaq projects;
- Access to environment variables of iDaq server.

Creating an ADD-ON requires a client satisfying particular requirements needed for a correct working of the entire server.

ADD-ONS REQUIREMENTS

This brief subchapter enlists the requirements being fundamental to create ADD-ONs: check them before starting.

- The panel must be set with **window behavior = floating** (eventual subpanels requiring data inputs by the user can be created through **window behavior = modal**);
- Clients **can't have splash pages**;
- Clients **can't have windows behavior = modal** at the start-up of user input panels;
- Clients **must use `isTerminated.vi`** function to end the execution indeed, in order to allow iDaq to close clients properly;
- Clients are **not enabled to ask user confirm before termination**, if `isTerminated.vi` returns TRUE value;
- **`Task` object can't be initialized by clients**: this function is supposed to be a server task;
- The client must contain **an iDaqTask control named `task`;**

As regards software requirements, the following list resumes the fundamental ones:

- iDaq for Windows 1.0.9.x or higher
- LabVIEW 2012;
- SCCT 2.6.x or higher;
- iDaq-DT 1.1.x or higher;

GETTING STARTED WITH ADD-ONS DEVELOPMENT

This chapter describes the necessary steps to develop your own ADD-ON. In particular, this paragraph explains about how to create a plug-in that in the end succeed in having the following features:

- Reading in a chart all the data coming from the first analog channel available from a selected data source;
- Enabling, therefore, to select a chosen data source among those available;
- Having two buttons: one for data view and one for freeze frame.

The complete project is available at the following URL: www.toolsforsmartminds.com

STEP ONE: STARTING PROGRAMMING ADD-ONS

First of all, launch LabVIEW and create a **new project named MyFirstADDON**: in the same way in this project create a **LabVIEW library** and name it MyFirstADDON. In the next image it is illustrated the project content and the files hierarchy.

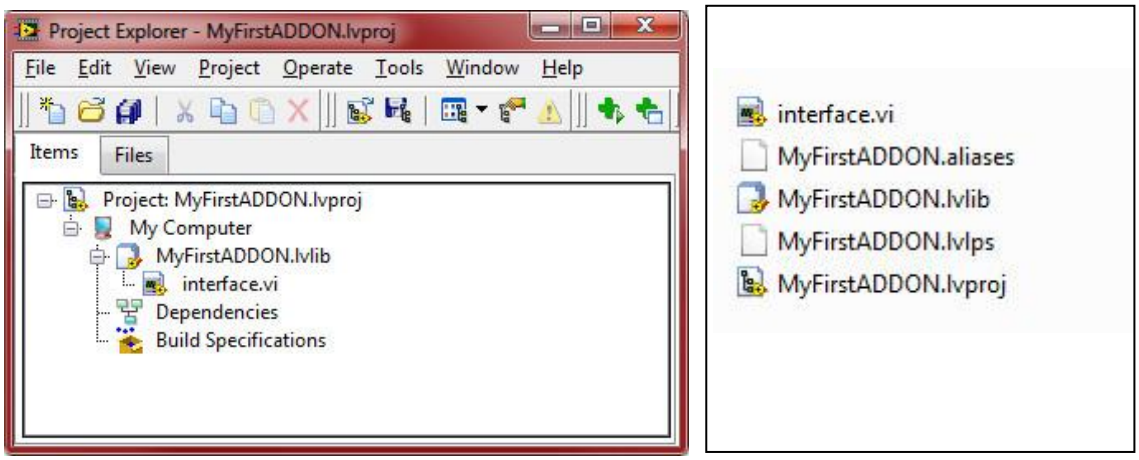

FIGURE 24 CREATING MYFIRSTADDON PROJECT AND FILES HIERARCHY

Now, create a new .vi file in the MyFirstADDON library and name it `interface.vi`. Combining LabVIEW palette with iDaq-DT functions allows you to prepare the `Connect.vi` function, to which it has to be connected a control named `task`.

The next figure represents punctually the connection between them:

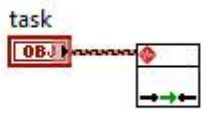

FIGURE 25 CONNECT.VI FUNCTION AND TASK CONTROL REPRESENTATION

NOTE:

ADD-ONs MUST contain an `iDaqTask` object named `task`, because iDaq needs it to interact with all the executing ADD-ONs. The initialization of this object is made by iDaq before ADD-ON execution.

iDaq, makes use of `iDaqTask` objects to terminate all of the executing ADD-ONs through the `terminate.vi` method and can understand whether the ADD-ON has been terminated by a user through the `isTerminated` method.

STEP TWO: CONNECTING TASK WITH TWO WHILE LOOPS

Connect the wire of `task` object with two while loops: the **first one**, for events handling, the **second one**, for data received displaying. Both while loops end when the `isTerminated.vi` function returns "TRUE". It is not necessary to use this programming scheme, but it is fundamental that when `isTerminated.vi` function returns a TRUE value, all of the active loops in an ADD-ON end. That is the only way in which iDaq can handle the different types of ADD-ONs contained in the iDaq work projects. If you want more details about iDaq work projects, we suggest you to take a look at the iDaq user manual at http://www.toolsforsmartminds.com/pdf/iDaq%20User%20Manual.pdf.

In the following figure it is represented the resulting code, as described before:

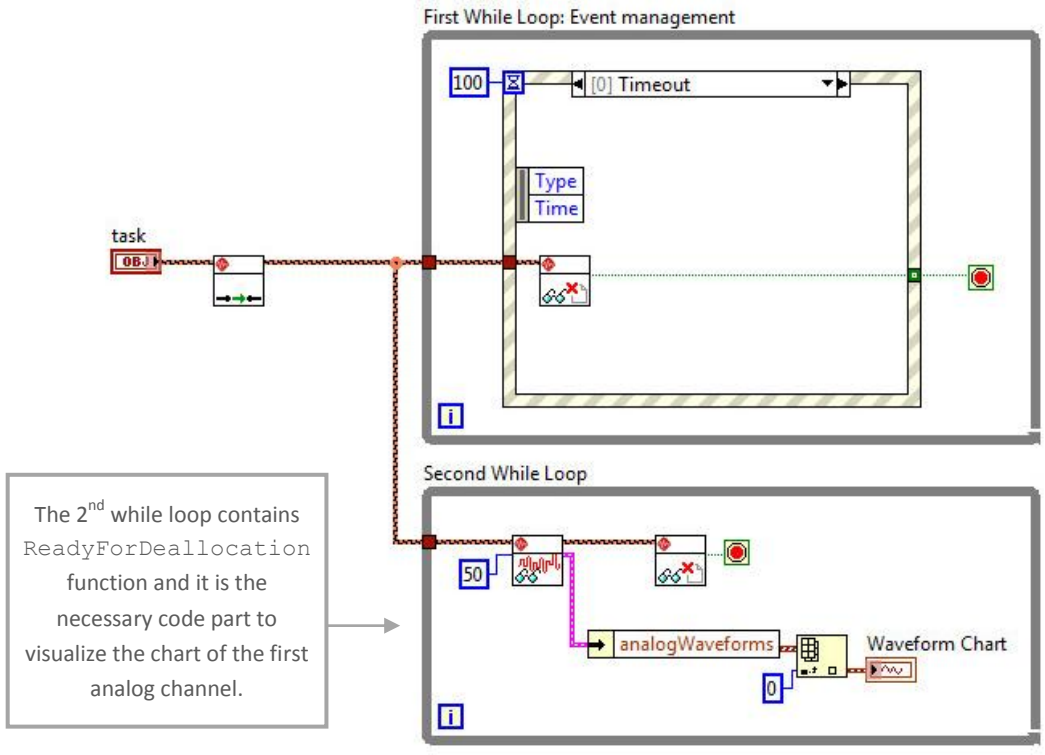

FIGURE 26 "CONNECTING CONNECT.VI WITH TWO WHILE LOOPS" SAMPLE CODE

STEP THREE: ADDING BUTTONS TO THE PANEL

Now, add three buttons to the panel: **play**, **pause** and **stop**; then create the events that execute the `play.vi` and `terminate.vi` functions within the events handling structure, as you can see in the next figure.

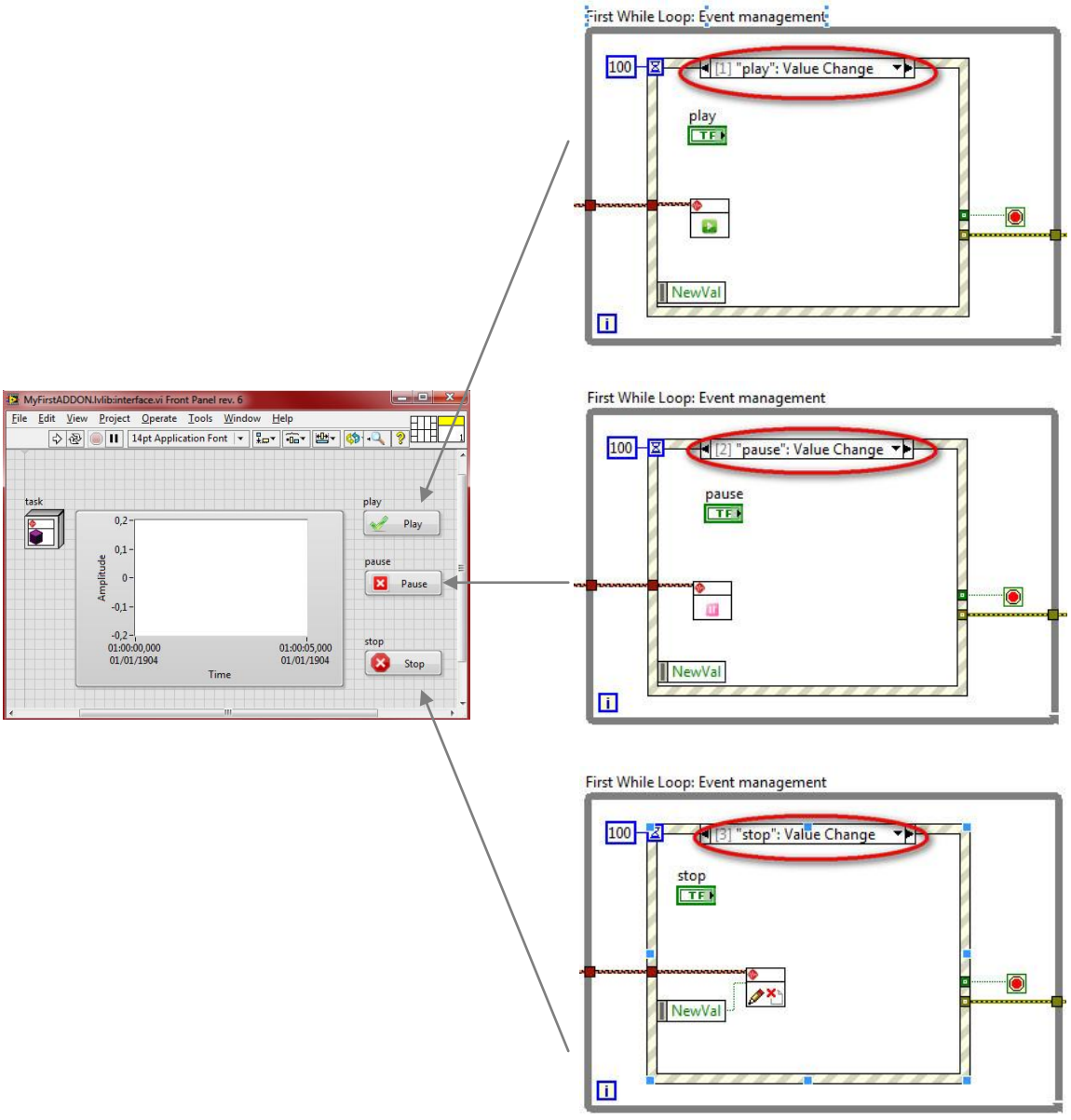

FIGURE 27 ADDING PLAY, PAUSE AND STOP BUTTONS TO THE PANEL

STEP FOUR: DISCONNECTION AND DEALLOCATION

Remember that at the end of while cycles it is fundamental to set the `disconnect.vi` and `ReadyForDeallocation.vi` functions in order to allow iDaq to remove ADD-ONs from memory in the right way.

NOTE:

After the execution of the ReadyForDeallocation.vi function, no other method can be executed on Task object, because from that point the ADD-ON could be deallocated by iDaq.

If disconnect.vi is not executed and ADD-ON is stopped, iDaq server is capable to recognize that the client is not more running and closes connection automatically. Using disconnect.vi is important to remove unused data structures and background tasks at client side.

The following figure illustrates the correct placing of the `disconnect.vi` and `ReadyForDeallocation.vi` functions at the end of the while cycles. The Error wire has been used to assure that both cycles are ended. Notice that task object doesn't need to be updated at the end of the while cycles.

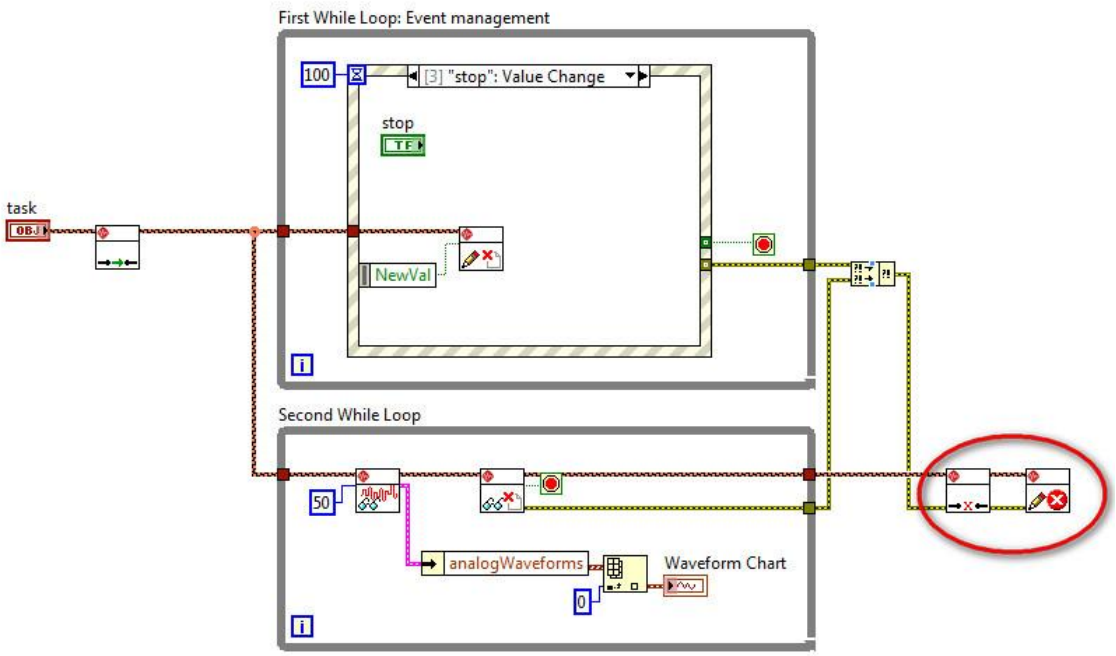

FIGURE 28 CORRECT PLACEMENT OF READYFORDEALLOCATION.VI FUNCTION

STEP FIVE: TESTING CODE

At this point, if you want to test the just created sample code, you can do it in the next way:

- Launch iDaq on a PC easily reachable from the computer you have used to create the code (as main hypothesis of the following paragraphs, iDaq is supposed to be executing on the same computer in which LabVIEW has been used.
- Verify the presence of at least one active data source in iDaq. In the project folder create a text file named `addon-configuration.INI` in which a section has to be created with the ADD-ON name and the keys values within it, indicated in the next figure:

```
[My First ADD-ON]
interfaceFileName = interface.vi
libraryFileName = MyFirstADDON
```

FIGURE 29 CREATING A SECTION WITH ADD-ON NAME AND KEYS VALUES

This file has to be saved in the same folder in which the LabVIEW project file (.lvproj) is placed.

- From LabVIEW, now, select **Menu Tool ▸ T4SM ▸ iDaq-DT ▸ ADD-ON Launcher**. ADD-ON launcher pops up (from now on this manual calls it "Launcher" for simplicity's sake) and allows to simulate iDaq's functionalities within LabVIEW. *Launcher* is a practical tool very useful during application debug.
  In this paragraph are illustrated only main Launcher functionalities: you can find more details in the "ADD-ON Launcher User Guide" at www.toolsforsmartminds.com.

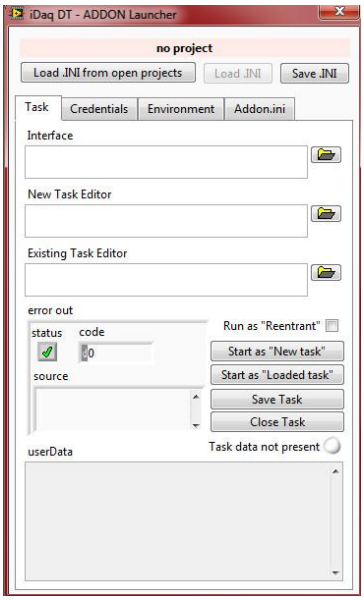

FIGURE 30 ADD-ON LAUNCHER WINDOW

If you haven't a credential file available to connect the ADD-ON to iDaq, use the **Credential ▸ Edit** button and type in either IP *Address* or the *name of the PC* where iDaq is executing, *TCP port* and *authentication password.*

It is possible to change the values used by iDaq (Port = 9000, Password = iDaq) through the iDaq's menu **Configuration ▸ Options ▸ Data Distribution tab**, but look at the iDaq User manual for more information (you can find the link in the APPENDIX A table). T

The next image represents how to get Credentials without Credential files available:

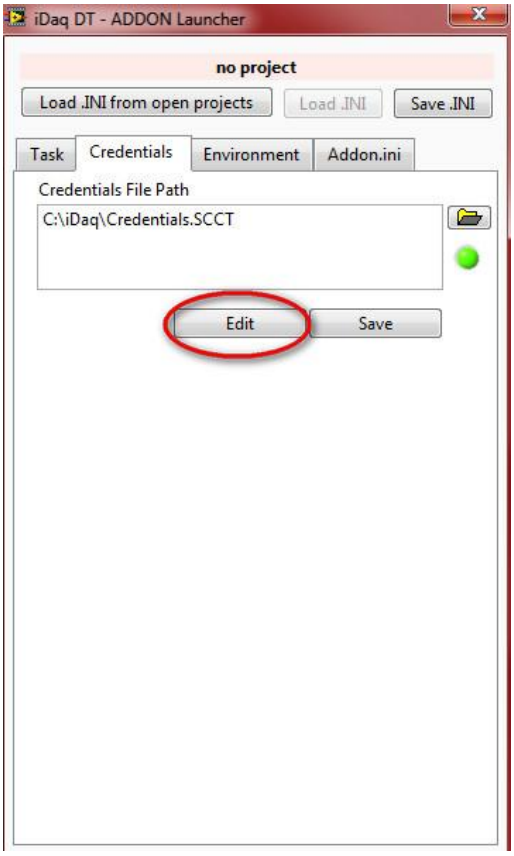

FIGURE 31 EDITING ADD-ONS CREDENTIALS

The example just mentioned uses the same iDaq credential file in the **c:\iDaq** folder. The next image illustrates the connection credentials window:

FIGURE 32 CONNECTION CREDENTIALS EDIT WINDOW.

When you are sure that the credentials file allows to connect with iDaq, click on the "`Load.INI from open projects`" button, so that Launcher transfer data in the `addon-configurator.INI` file created before.

Click on the `start as "New Task"` button and the just created .vi start being executed, as the next image shows:

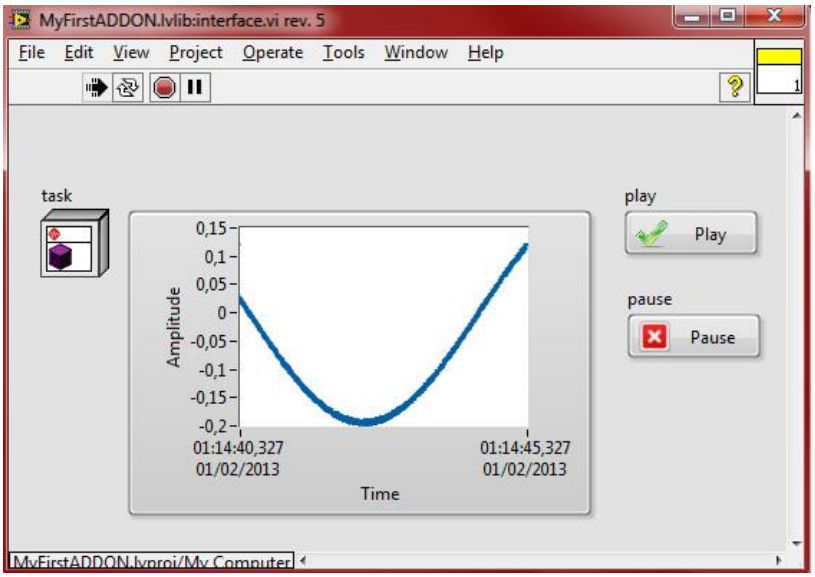

FIGURE 33 EXECUTING ADD-ON

APPENDIX A: USEFUL LINKS

In this appendix we make a quick resuming list of the most useful links you have to look up to get the best results from iDaq-DT.

Links	About links
www.toolsforsmartminds.com	For any information or support about iDaq
www.toolsforsmartminds.com/products/programmable_data_logger.php	Download here iDaq-DT
https://lumen.ni.com/nicif/us/evaltlktvipm/content.xhtml	Download here VIP manager
www.ni.com	Download here the drivers for NI devices.
http://www.toolsforsmartminds.com/products/SCCT.php	Download here SCCT
http://www.toolsforsmartminds.com/pdf/SCCT%20User%20Manual.pdf	Look up here for SCCT user manual
http://www.toolsforsmartminds.com/pdf/iDaq%20User%20Manual.pdf	Look up here for iDaq user manual
www.toolsforsmartminds.com	Look up here for stand-alone application sample code
www.toolsforsmartminds.com	Look up here for ADD-ON sample code.

FIGURE INDEX

FIGURE 1 IDAQ-DT INFORMATION PACKAGE DURING GUIDED INSTALLATION. 11

FIGURE 2 NECESSARY CODE FOR SERVER CONNECTION 12

FIGURE 3 CONNECTION PARAMETERS EDITOR 13

FIGURE 4 CONNECTING IDAQTASK WITH CONNECT.VI 13

FIGURE 5 GETTING DATA SOURCE INFORMATION WITH READ SOURCELIST.VI FUNCTION 14

FIGURE 7 DATA SOURCE SELECTION 14

FIGURE 5 ELABORATING CLUSTER WHEN BOOLEAN FOUND = TRUE 14

FIGURE 9 DISCONNECT FUNCTION 15

FIGURE 7 SOURCE SELECTION SAMPLE CODE AND DIALOG BOX 15

FIGURE 10 REAL TIME EXTRACTING PACKETS IN WHILE LOOP 16

FIGURE 11 STARTING DATA SOURCE TRANSMISSION WITH PLAY.VI 17

FIGURE 12 STOPPING DATA SOURCE TRANSMISSION WITH PAUSE.VI 17

FIGURE 12 DATA TRANSMISSION PROCESS FROM DATA SOURCE TO CLIENT, ... 17

FIGURE 14 SAMPLE APPLICATION OVERVIEW 18

FIGURE 15 SAMPLE CODE TO INITIALIZE IDAQTASK AND SERVER CONNECTION 19

FIGURE 16 DATA ACQUISITION WHILE LOOP 19

FIGURE 17 SAMPLE CODE OF AN EVENT HANDLING STOP BUTTON 20

FIGURE 18 SAMPLE CODE OF AN EVENT HANDLING SOURCE SELECTION 20

FIGURE 19 RESULTING CODE OF A SAMPLE STAND-ALONE APPLICATION 21

FIGURE 20 SERVER-CLIENT DATA TRANSMISSION PROCESS DIAGRAM 22

FIGURE 21 SAMPLE CODE FOR FILTERDATA.VI IMPLEMENTATION 23

FIGURE 22 DELEGATING FILTERID CONTROL TO IDAQTASK 24

FIGURE 23 SEQUENCE OF PACKET EXCHANGED BY SERVER AND CLIENT WITH FILTERDATA.VI 24

FIGURE 25 CONNECT.VI FUNCTION AND TASK CONTROL REPRESENTATION 27

FIGURE 23 CREATING MYFIRSTADDON PROJECT AND FILES HIERARCHY 27

FIGURE 25 "CONNECTING CONNECT.VI WITH TWO WHILE LOOPS" SAMPLE CODE 28

FIGURE 26 ADDING PLAY, PAUSE AND STOP BUTTONS TO THE PANEL 29

FIGURE 28 CORRECT PLACEMENT OF READYFORDEALLOCATION.VI FUNCTION 30

FIGURE 29 CREATING A SECTION WITH ADD-ON NAME AND KEYS VALUES 31

FIGURE 30 ADD-ON LAUNCHER WINDOW 31

FIGURE 31 EDITING ADD-ONS CREDENTIALS 32

FIGURE 32 CONNECTION CREDENTIALS EDIT WINDOW. 33

FIGURE 33 EXECUTING ADD-ON 33

INDEX

C

Close connection 15

Connection credentials..... 12

Creating ADD-ONs

ADD-ON Launcher 30

addon-configuration.INI file 30

addon-configurator.INI file..... 32

ADD-ONs credentials 31

LabVIEW project file 30

Load.INI from open projects button..... 32

MyFirstADDON library..... 26

ReadyForDeallocation.vi..... 29

start as “New Task” button 32

D

Data server 12

E

Errors

Server connection error 13

F

Functions

Changesource.vi 14

connect.vi 13; 14; 26

disconnect.vi 15

FilterData.vi 23

interface.vi..... 26

isTerminated.vi..... 19; 25

Read streams.vi..... 16; 19; 23

ReadyForDeallocation..... 27; 29

selectSourceDialog.vi 20

sourceList.vi 14

sourceSelectionDialog.vi..... 20

startTask.vi..... 12

terminate.vi 20

I

iDaq user manual 22

M

mono-source client 15

O

Objects

iDaqTask..... 12; 13; 15; 19; 20; 23; 25

Open connection..... 12; 13

Optimizing CPU 22

P

Parameters

credentialFilePath 12

error out.source..... 13

FilterID 23

found boolean 14

R

Requirements

ADD-ONs requirements 25

Minimum hardware requirements 10

Minimum software requirements 10

S

Saving band 22

Server address 13

Server password 13

Server port 13

Stop button 18; 20

Stop data communication 17

U

Using a subset of analog channels 23

W

While loop 16; 18; 19; 27